Software Aging and Multifractality of Memory Resources

January 8, 2003

Abstract

We investigate the dynamics of monitored memory resource utilizations in an operating system under stress using quantitative methods of fractal analysis. In the experiments, we recorded the time series representing various memory related parameters of the operating system. We observed that parameters demonstrate clear multifractal behavior. The degree of fractality of these time series tends to increase as the system workload increases. We conjecture that the Hölder exponent that measures the local rate of fractality may be used as a quantitative measure of software aging. We propose a simple proactive computer crash avoidance strategy based on the online fractal analysis of system memory resource observations.

1 Introduction

1.1 Software Aging

A phenomenon known as "software aging" has been identified and described in recent literature [22, 19, 17]. It is important to distinguish between the two definitions of software aging discussed in the literature. The first use of the term was introduced by David Parnas [14] and refers to the degradation of a software system performance over many years as maintenance of the system introduces significant changes and/or as the requirements of the organization change. The second use of the term, the one we are concerned with here, refers to the performance degradation of a software system over a period of hours, days or months, as errors accumulate while the system is running. This use of the term "software aging" is applicable mainly to software systems that are designed to run for long periods of time, such as a server in a client-server application, or a control system on a long-term space mission.

All computer users are familiar with the phenomena of operating system crashes and software freezes. These unexpected events usually result in a loss of time and, eventually, data. The first and most important way to counteract system crashes is to design and build better software. Large software systems, however, are extraordinarily complex entities, and achieving perfection in their design and implementation is unlikely if not impossible. The next line of defense is to take proactive measures to anticipate crashes so that data can be saved and the system can be shut down, cleaned, and restarted in a rejuvenated state. This process has been described as "software rejuvenation" [22, 18, 16].

The causes of software aging are the accumulation of numerical rounding errors, the corruption of data, the exhaustion of operating system resources, unreleased file locks, memory leaks, and other similar minor malfunctions [6]. Errors and failures are due to faults described by Gray [4] and Huang et al. [22] as *Heisenbugs*. The activation of these faults is non-deterministic and they could remain inactive for a long time. *Bohr bugs*, on the other hand, are deterministic and predictably lead to a software failure. *Heisenbugs* are difficult if not impossible to detect by system testing due to their non-deterministic character. Indeed, the term was first used to describe those bugs that cease to occur whenever debugging or tracing flags are used during software compilation.

Since these errors are very difficult to anticipate directly, we must look for other ways of predicting when a system is in a vulnerable state. Vaidyanathan and Trivedi [6] propose a semi-Markov reward model based on system workload and resource usage to estimate the time of failure of a system. However, the data they collected tend to fluctuate a great deal from the supposed linear trends, resulting in prohibitively wide confidence intervals. The dynamics of memory resources appears to be heavily irregular and nonlinear.

In the last decade there have been successful attempts by many researchers to apply fractal analysis to model and study highly irregular signals arising in various fields of natural sciences and engineering (see [20]). Our hypotheses have been that

1. the temporal dynamics of memory-related resources of an operating system has multifractal structure;
2. analysis of the multifractal patterns in time series of the system resources can be used for determining whether an operating system has begun to age and is likely to crash.

The results presented in the paper provide initial support for the above hypotheses.

1.2 The Fractal Approach to Software Rejuvenation

Fractal geometry is a relatively new area within mathematics, having been developed in the 1960s and 70s by Benoit Mandelbrot [9], which is continually proved useful in modeling various complex, chaotic phenomena of nature. In the recent years it has been discovered that many objects of human derivation such as network traffic patterns [21] and the stock market [15, 8] display fractal characteristics. Fractal analysis has been employed with success in the analysis and synthesis of speech signals [10, 2], in the analysis of human heartbeats [13, 12], and in image compression algorithms [1, 5]. The goal of our study is to establish that the time series of resource use in operating systems display fractal behavior, whose quantitative characteristics vary with time. Such behavior is called multifractal. We also conjecture that the patterns of the multifractal behavior of these signals contain information about the "age" of the operating system and can be exploited to predict the best times to engage in the preventive maintenance of the operating system. The results reported here appear to support this hypothesis and indicate a possible approach to solving the problem.

1.3 Structure of the Paper

The remaining sections of the paper will be devoted to explaining the mathematical theory that underlies our approach, discussing the operating system resources we measured and their fractal characteristics, explaining the design of the experiment, suggesting possible methods for developing a model for the detection of software aging and the prediction of imminent operating system crashes, and suggesting directions for further work.

2 Multifractality in Functions and Signals

A function or a signal is usually referred to as fractal if its graph displays such fractal characteristics (see e.g. [3]) as (local) self-similarity, irregularity, fine structure, and fractional dimension. When viewed as a random process,. A function that displays variable local scaling at different points is known as multifractal. Multifractal analysis deals with the description of the singularity structure of functions, signals or measures (see [23]).

The basic numerical characteristics used in the multifractal analysis is the so-called Hölder exponent which has lately become a popular signal processing tool among scientists in various fields that deal with irregular and rapidly oscillating data (ee e.g. [13] for an example of application of Hölder exponent in the analysis of the heart beat signal). Hölder exponent which is discussed in more detail in the next subsection, captures the local multifractal information. The global multifractality is described through geometrical or statistical distribution of the Hölder exponent i.e. the multifractal spectrum. In this study we have focused on the variation of the local fractality of oprating system resources, and therefore Hölder exponent has been our primary quantitative tool. Another reason for using Hölder exponent in our context is that it is relatively easy to estimate in an online experiment. However, we believe that using methods of the global multifractal analysis may yield additional insight into the dynamics of the resource exhaustion, and plan to address this issue in our future publications.

2.1 The Hölder Exponent

The Hölder exponent of a function is a local characteristic calculated at every point where the function is defined and reflecting the decay rate of the amplitude of the functions fluctuation in the neighborhood of

that point as the size of the neighborhood shrinks to zero. A highly irregular and chaotic point in a function is characterized by a lower value of Hölder exponent and a smoother portion of a function will have higher Hölder exponent. A function may have different Hölder exponents at different points due to a variation in the local degree of fractality, in which case it is referred to as multifractal.

A function f defined in a neighborhood of a point t_0 has a Hölder exponent α at t_0 if and only if (see [7]):

1. For every real $\gamma < \alpha$:

$$\lim_{h \to 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = 0$$

and

2. If $\alpha < +\infty$, for every real $\gamma > \alpha$:

$$\limsup_{h \to 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = +\infty$$

where P is a polynomial whose degree is less than or equal to α.

We will denote the Hölder exponent of a function at t_0 as $\mathcal{H}_f(t_0)$. It is easy to see that $\mathcal{H}_f(t) \geq 1$ if and only if the function $f(t)$ is differentiable (locally smooth), at t_0. For instance, in the extreme case when the function is (locally) constant near t the definition of the Hölder exponent clearly implies that for such a function $\mathcal{H}_f(t) = \infty$ (since $\log(0) = -\infty$). A non-constant linear function $f(t) = at + b$ with $a \neq 0$ has Hölder exponent 1.

The value of Hölder exponent $\mathcal{H}_f(t) = \alpha$ strictly below 1 indicates fractality: it implies that the function is not differentiable at the given point and the amplitude of its oscillations within ϵ-neighborhood of the point scales as ϵ^α for $\epsilon \to \infty$. Different values of Hölder exponent indicate different "degrees of fractality". Classical fractal functions are strictly self-similar, i.e., their singularities have the same scaling at every point. Such is the Weierstrass function, described in the next subsection.

In the most of our experimentally collected data sets, however, the Hölder exponent stays within the range $(0, 1)$ which characterizes a non-differentiable (or fractal) signal.

2.2 Calculating the Hölder Exponent

Véhel and Daoudi have shown [20] that the Hölder exponent at a point t of a function $f(t)$ can be expressed by the formula

$$\mathcal{H}_f(t) = \liminf_{h \to 0} \frac{\log(|f(t + h) - f(t)|)}{\log(|h|)}. \tag{1}$$

The definitions and formulas for the Hölder exponent given above assume f to be a function of a continuous variable t (time). In the experimental context however we deal with a discrete signal, and therefore the numerical computation of Hölder exponent presents a problem. Even though algorithms for calculating the Hölder exponent of a time series have been developed and used, they were not suitable for our purposes, since our goal was to monitor the parameters of an operating system and compute their Hölder exponent in real time. The existing algorithms are based either on the IFS approximation or the wavelet expansion of the signal and, therefore, require the knowledge of the entire time series in order to estimate the Hölder exponent at any point. This feature renders them inconvenient for the task of on-line Hölder exponent estimation. We have developed an algorithm for on-line estimation of the Hölder exponent of a signal based on equation 1, the details of which are given in the Appendix.

We tested our algorithm for calculating the Hölder exponent by running it on so-called generalized Weierstrass functions [23] whose Hölder exponent can be predefined and then compared with the results obtained using the numerical algorithm. The generalized Weierstrass function is defined by the formula

$$f(t) = \sum_{k=0}^{\infty} 3^{-ks(t)} \sin(3^k t),$$

where $s(t)$ is the seed function ranging between 0 and 1 (see e.g. [7]). It is known that $s(t) = \mathcal{H}_f(t)$ for all t.

Figure 1 shows the plot of our Hölder exponent estimation compared to the predefined Hölder exponent $s(t)$ for the case when $s(t) = |\sin(5\pi t)|$, the generalized Weierstrass function with its Hölder exponent specified at every point by $s(t)$ is shown next to the seed function, and our estimation of the Hölder exponent along with an alternative estimate of the Hölder exponent obtained by a wavelet based algorithm used by [7], are shown in the bottom half of the figure.

Our algorithm demonstrated consistently high precision (consistently exceeding the presision achieved by the Daoudis method) for several test functions constructed according to the above formula.

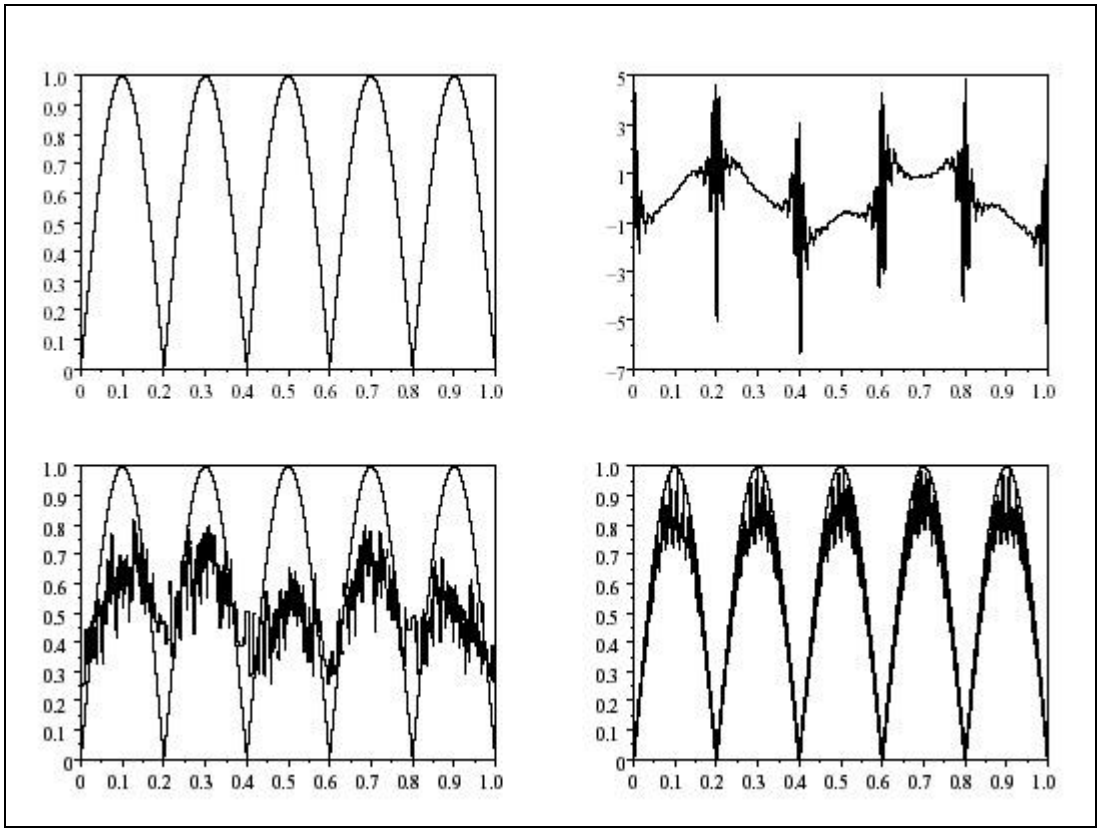

Figure 1: Our estimation of the Hölder exponent, on the lower right, compared with Dauodi's estimation on the lower left. The plot in the upper left is the plot of the seed function, and the plot in the upper right is the resulting Generalized Weierstrass function.

3 Fractality of Operating System Resources

Current operating systems have a great number of parameters that can be monitored during system operation. Not all of them, however, can be viewed as interesting or even meaningful for the kind of fractal data analysis we conduct.

It turns out that the majority of data collected by standard operating system data collection processes, viewed as mathematical objects, behave more like *measures*, not functions, i.e., they represent a *density* of certain flow of (random) events, not a quantity that describes the internal state of the operating system and evolves as a continuous function of time. Examples of the former type of parameters (*measures*) are the number of page requests per second, the number of system calls per second, the number of interrupts per second, and so on. The dynamics of this class of parameters is undoubtedly interesting and we expect them to also display fractal behavior. However, the machinery of the fractal analysis applicable to measures

differs considerably from the one applicable to functions. In this paper we are concerned with the latter, while planning to present a study of the former in future publications.

The most easily identifiable class of operating system parameters that represent a function (not a measure) consists of the parameters related to the memory use. These are the parameters that we focused on in our experiments.

Our first series of experiments in collecting and studying operating system data focused on a Unix server in a large academic computing environment. The server has 256 MB of RAM, dual 333 MHz processors, runs SunOS 5.5.1, and it usually experiences heavy computational workloads. Monitored parameter values were collected once per second using *sar* (system activity reporter) utility. We concentrated on the following six parameters of time-series data:

1. **sml_mem** - the amount of memory the kernel memory allocator has reserved for small requests.
2. **sml_alloc** - the amount of memory allocated to satisfy small requests.
3. **lg_mem** - the amount of memory the kernel memory allocator has reserved for large requests.
4. **lg_alloc** - the amount of memory allocated to satisfy large requests.
5. **freemem** - average pages available to user processes.
6. **freeswap** - disk blocks available for page swapping.

We found that two parameters displayed marked fractal characteristics, a couple were less fractal, and the two parameters dealing with the amount of reserved memory (**sml_mem** and **lg_mem**) demonstrated quite smooth behavior, with no obvious indication of fractality.

Figure 2 shows a plot of the **freemem** parameter with a plot of its Hölder exponent beneath it. In this case, the free memory resource displayed little fractality for the first 4900 seconds or so, and the bulk of the Hölder exponent values for that time period fall between 0.6 and 1. This is followed by an outburst of chaotic behavior, which is captured by the Hölder exponent as it falls to between 0.3 and 0.6. When the period of fractality is over, the Hölder exponent rises again.

A more intriguing interplay of data and Hölder exponent is seen in Figure 3. In this case, a sharp drop in the **lg_alloc** parameter is immediately preceded (by about 30 seconds) by an increased level of fractality (reflected by the drop in Hölder exponent). If this increase in fractality is in fact a predictor of the coming drop in the availability of the resource, then the shift in the Hölder exponent could be used to give an alarm that a change in system performance may be imminent. Measuring the **lg_alloc** parameter directly in this case would be very useful since, upon the onset of fractality, the resource does not display a very pronounced change in magnitude (in which case it would be easy to detect without resorting to fractal analysis). The Hölder exponent displays a marked change in magnitude, however, thereby enabling detection of the increased volatility in the original resource.

None of the data sets we collected from this server actually preceded a system crash. The monitored server did not crash during the experiment, the period exceeding four months of data collection. So, in order to test our hypothesis that an approaching system crash could be anticipated and prevented through fractal analysis of a system's memory use, we set up another experiment specifically designed to generate artificialy high workloads with known aging effects and observe computer crashes. In addition, we needed to more accurately model the behavior of the system and come up with an algorithm that would issue an alarm when an operating system crash becames likely. The data resulting from such an experiment, along with certain observations and conclusions derived from it, is discussed in the next two sections.

4 Design of Experiment

4.1 Setup

Our "crash experiment" used the 'System Stress for Windows NT 4.0 and Windows 2000' software that is included in a subscription to the Microsoft Developer Network (MSDN). Two computers were set up next

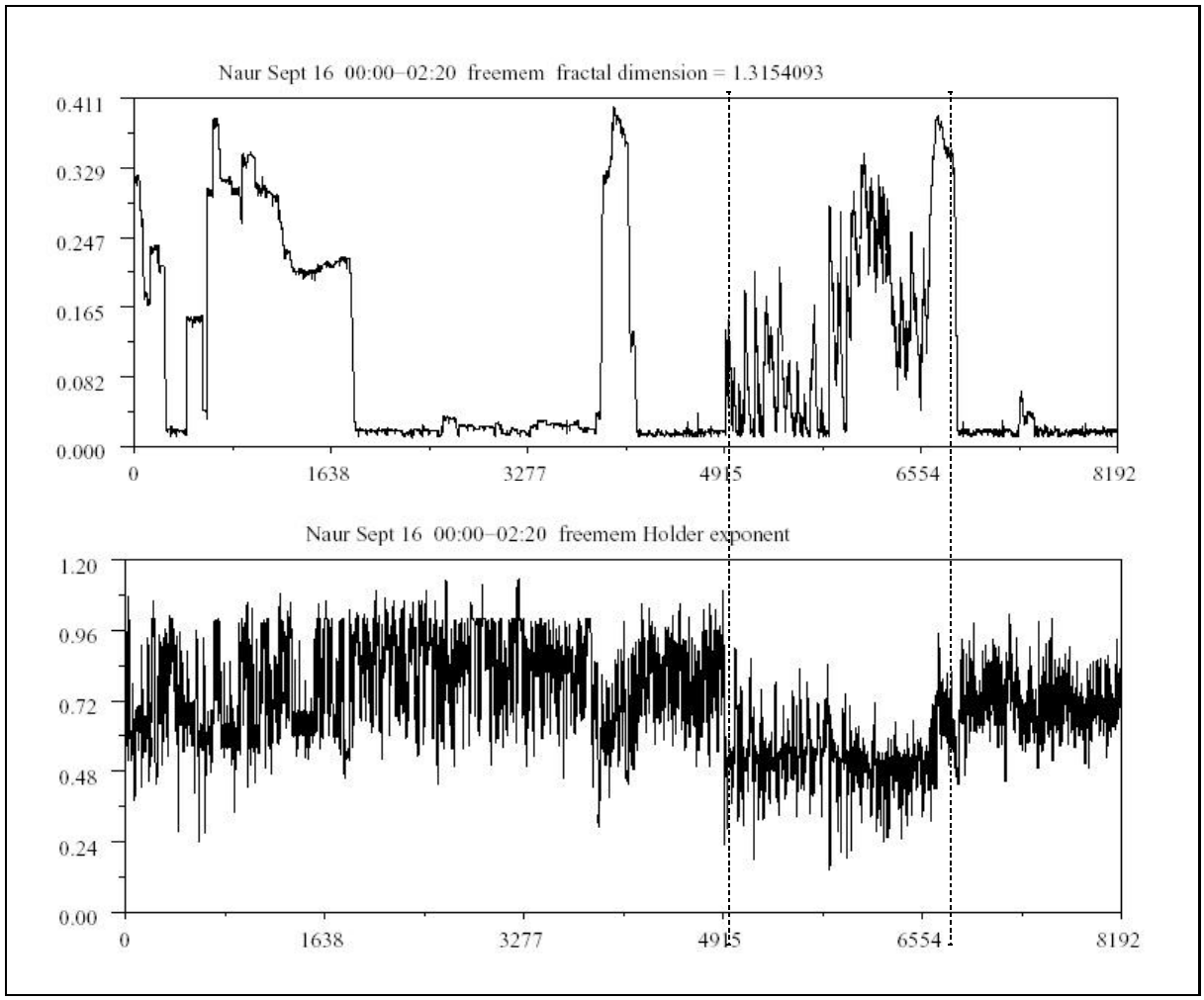

Figure 2: 8192 seconds of the freemem parameter from the Naur experiment along with its Hölder exponent .

to each other and networked together with a crossover cable, thus forming a LAN of only the two machines. The System Stress software was installed on one computer (a Dell machine with 600 MHz processor, 64 MB of RAM, running Windows 2000) and configured to contact the other computer with service requests. The computer that was subject to the stress was a home-grown machine with a 750 MHz AMD processor, 256 MB of RAM, and running Windows 2000.

The System Stress utility gives the user a large number of stress programs that may be used in any particular trial. The selection of which additional stress program to add to the machine was made by a few lines of java code that output a random number. Every few minutes the administrator of the experiment would randomly generate a short list of additional programs and add them to the current stress test. Once a large enough number of programs were added to the stress test, the stress would reach the level of exhaustion of operating system resources necessary for causing the system crash.

Prior to starting the stress utility, a performance log would be set up to record data every second for the duration on the trial. The utility used for the collection of data was the *"Performance Logs and Alerts"* utility available in the Windows 2000 Operating System. During the experiment we initially collected data on 68 different parameters of the operating system, but winnowed this number down to 20 for our subsequent analysis. This number was eventually reduced to the three. The crash of the operating system naturally disrupted the collection of data, but the process was robust enough to collect data until the penultimate

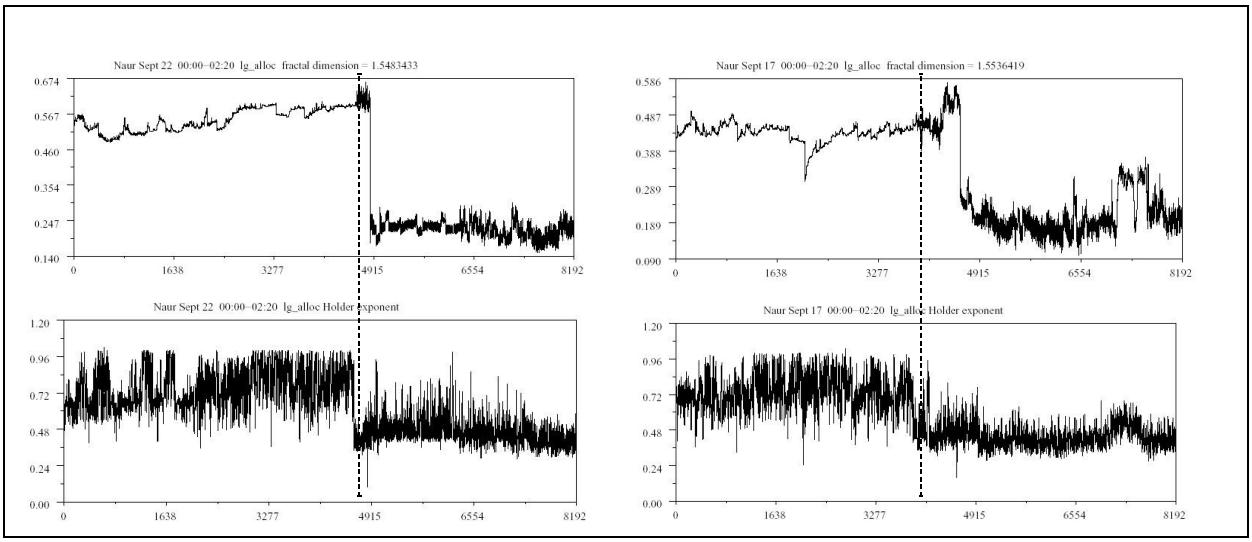

Figure 3: Onset of Fractality Precipitates Fall in Resource Availability (lg_alloc).

second before the crash. The experiment was run 20 times, with a crash sometimes occuring within the first 5 minutes of the experiment, and sometimes occurring after more than a day of the operation.

4.2 The Data

The data we collected and analyzed confirmed our hypothesis that an increase in fractality tends to precede a crash. Figure 4 shows the plots of the three parameters we monitored with the Hölder exponent of the corresponding three-dimensional time series plotted below them.

5 Fractal Aging: Analyzing Hölder Exponent Plots

Even a cursory glance at the Hölder exponent plots depicting the dynamics of the monitored operating system resources shows that, typically, a decreasing trend in the value of Hölder exponent (i.e. increase in fractality) of the memory resource dynamics is observed as the stress on the system increases before the system crashes (see Hölder exponenttime series from eight different experimental runs in Figure 5).

The collected data reinforced our belief that *the process of resource exhaustion is quantitatively manifested in the decrease of the value of Hölder exponent of the system's memory resources considered as functions of time.* However, to be of any practical value, this intuitive observation has to be converted into a precise quantitative indicator. This problem required further investigation, both experimental and theoretical. Here we present only the first step in this direction, which suggests the possibility of developing an 'early crash warning' algorithm for an operating system based on the on-line multifractal analysis of its resource data. Such an algorithm can be used to build an efficient automated software rejuvenation tool.

We started by selecting a small group of resources suitable for the analysis. We based our choice on the following selection criteria:

1. The resource should represent "continuous" quantity, and it should not represent "per unit of time" measurements. This criterion disqualifies such resources as system_driver_total_bytes, because of their discrete, discontinuous nature, as well as system_calls_per_sec and such, since they represent quantities measured per unit of time (second). Remark: It is worth noting that the per-unit-of-time quantities may be of considerable interest for the multi-fractal analysis. However, they should be treated as multi-fractal measures, as opposed to multi-fractal signals. Therefore the computational methods involved in it would be rather different. We will address the multi-fractal analysis of this type of resources in our future studies.

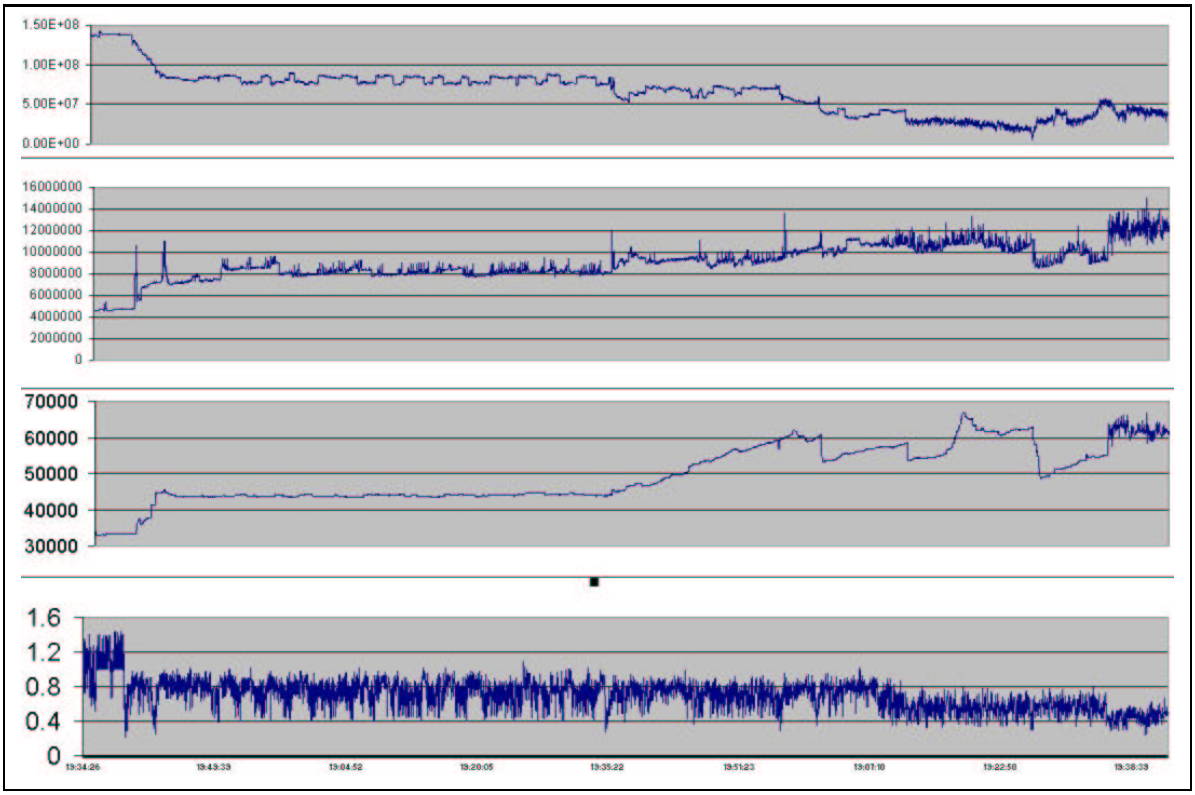

Figure 4: Available Bytes, Pool Paged Allocs, System Cache Resident Bytes, and their multidimensional Hölder exponent .

2. The resources whose plots exhibit flat or smooth behavior or consist of several intervals of such behavior are not suitable for the fractal analysis. In our experiments the resources system_code_total_bytes and system_driver_resident_bytes demonstrated such behavior and, therefore, were excluded from the monitoring effort.

3. The group of resources selected for analysis should *not* have high mutual correlations.

Proceeding in accordance with the above guidelines, we have selected the following three system parameters:

1. available_bytes,

2. pool_paged_allocs,

3. system_cache_resident_bytes.

Please note that the choice is not uniquely determined by the guidelines. All these parameters represent various kinds of memory resources, they demonstrate fractal behavior and do not have too high mutual correlations. The following table gives the correlations for the three parameters we have chosen:

	avail_bytes	pool_page_allocs	sys_cache_res_bytes
avail_bytes	1.0	-0.66	-0.85
pool_page_allocs	-0.66	1.0	0.5
sys_cache_res_bytes	-0.85	0.5	1.0

All the (non-diagonal) mutual correlation values have absolute value ≤ 0.85. We wanted to incorporate into our analysis the fractal behavior of all three of these parameters. To this end, we considered the three parameters as a 3-dimensional memory resource vector. We sample this vector as it changes in time resulting in a three-dimensional time series, to which we apply the multi-fractal analysis. The analysis is performed on-line, with its Hölder exponent computed at every point of observation. The formula for the Hölder exponent is slightly modified in the case of a multidimensional function:

$$\mathcal{H}_f(t) = \liminf_{h \to 0} \frac{\log(||f(t+h) - f(t)||)}{\log(|h|)}.$$

The algorithm we used to estimate the Hölder exponent of the multidimensional time series is the same as described in Section 2.2, only instead of the difference between two points, we use the Euclidean distance.

Figure 5 shows the plots of the Hölder exponent of the *3-dimensional memory resource vector* in 8 experiments ending with the system crashing. Though the Hölder exponent series obtained are rather noisy (this is typical for experimentally estimated Hölder exponent values), certain common patterns can be distinguished.

Even by a mere visual inspection of the plots, one can observe that the plots have *long intervals during which the Hölder exponent fluctuates around a certain level*. This quasi-constant behavior is sometimes (infrequently) interrupted by moments of abrupt significant changes (mostly drops) in the average value of Hölder exponent. We call a sudden sustained drop in the average value of Hölder exponent a *fractal breakdown*. Most of our experimental plots (twelve out of the fifteen data series we analized) contain exactly two such breakdowns. Based on this observation we make the following

Conjecture. The *second fractal breakdown* observed during the experimental system observation signals a dangerous level of resource exhaustion, but leaves enough time to shut the system down before the crash occurs.

6 Detecting the Fractal Breakdowns

In order to validate the above conjecture and implementing the emergency shutdown strategy based on it we faced the following problems:

1. *Detecting fractal breakdown.* We view the problem as that of detecting a change in the mean value of a noisy time series (see e.g. [11]). Here the time series in question is the one formed by values of the Hölder exponent of the memory resource vector computed during the experiment. We model the behavior of the Hölder exponent as a piece-wise constant function of time with a white Gaussian noise added to it. In the following subsection we describe the application of the classical Shewhart change detection algorithm [11] to solve the problem of detecting the second fractal breakdown in our time series.

2. *Determining the optimal shutdown time after the second breakdown has been detected.* In order to infer the optimal strategy for choosing shut-down time to, on one hand, let the system run as long as possible, while, on the other hand, not allowing it to crash before the shutdown, we plan to study the statistics of the time intervals between the second breakdown and the crash. This remains the topic of the future research and it is not reported herein.

6.1 Using Shewhart Algorithm for Fractal Breakdown Detectionion

Though numerous good change detection algorithms exist, most of them presuppose the a priori knowledge of the main parameters of the post-change signal. This is not the case in our situation where the change detection needs to be performed in real time. We used the classical *Shewhart control charts* algorithm. However, for the reason mentioned above, we needed to modify it so that, unlike in the classical situation, the mean and the variance of the signal are estimated on-line instead of being known *a priori*.

The details of the algorithm we used to detect the sudden drops in the Hölder exponent signal are given in Appendix 2.

Figure 6 shows four sample plots of the Hölder exponent of the combined (3-dimensional) resource vector, where the two sudden drops in the mean value of Hölder exponent (fractal breakdowns) occur before the system crashes. The moments of the 1^{st} and the 2^{nd} fractal breakdowns have been detected using the Shewhart Control Charts algorithm, as described in Appendix 2. In the charts shown in Figure 6 the time elapsing between the 2^{nd} fractal breakdow and the moment of crash failure varies from three minutes to over thirty minutes. This situation is characteristic for our entire collection of data sets. In other words, the time between the 2^{nd} drop in Hölder exponent and the crash of the system varies widely, thus posing a difficult problem of developing an *optimal* software rejuvenation strategy based on the fractal breakdow events. However, we consider our current result of successfully predicting the occurrence of the upcoming system crash in 12 out of 15 experiments a success and a sound basis for continued research.

7 Conclusion

This paper presents an investigation of the dynamics of memory resource utilizations in an operating system using quantitative methods of fractal analysis. To the best of our knowledge, this is the first study attempting to utilize the principles of fractal analysis to performance analysis of computer systems.

In the experiments, we recorded the time series representing various memory related parameters of the operating system. We report the observation that several operating system parameters demonstrated clear multifractal behavior. Interestingly, the degree of fractality of these time series tends to increase as the system workload increases. Therefore, we conjecture that the Hoelder exponent, which measures the local rate of fractality, may be used as a quantitative measure of operating system resource exhausion. We developed a simple and fast proactive computer crash avoidance strategy based on the online fractal analysis of system memory resource observations. The described algorithm succesfully predicted upcoming system crashes in 80% of our experiments.

While promissing, the results of this paper should be considered as preliminary and their general applicability, at this point in time, is not guaranteed. It is easy to note that our experimental setup was rather simplistic. In the local area network, the computer under observation was exposed to increasing workloads only. It is unclear whether the same simplistic fractal functions with few (in the specific case two breakdown points) would result from experiments with varying workloads (increasing as well as decreasing). Furthermore, while we reduced the number of monitored variables to only three, the impact of monitoring to system

performance should not have been ignored. We noticed differences in system performance in experiments with a different number of operating system variables being monitored.

In summary, while exciting and promissing, our current result indicate the need for further experimentation and improved modeling. However, we do believe that the demonstrated suitability of multifractal analysis methods cannot be ignored and will play an increasingly important role in computer system performance modeling.

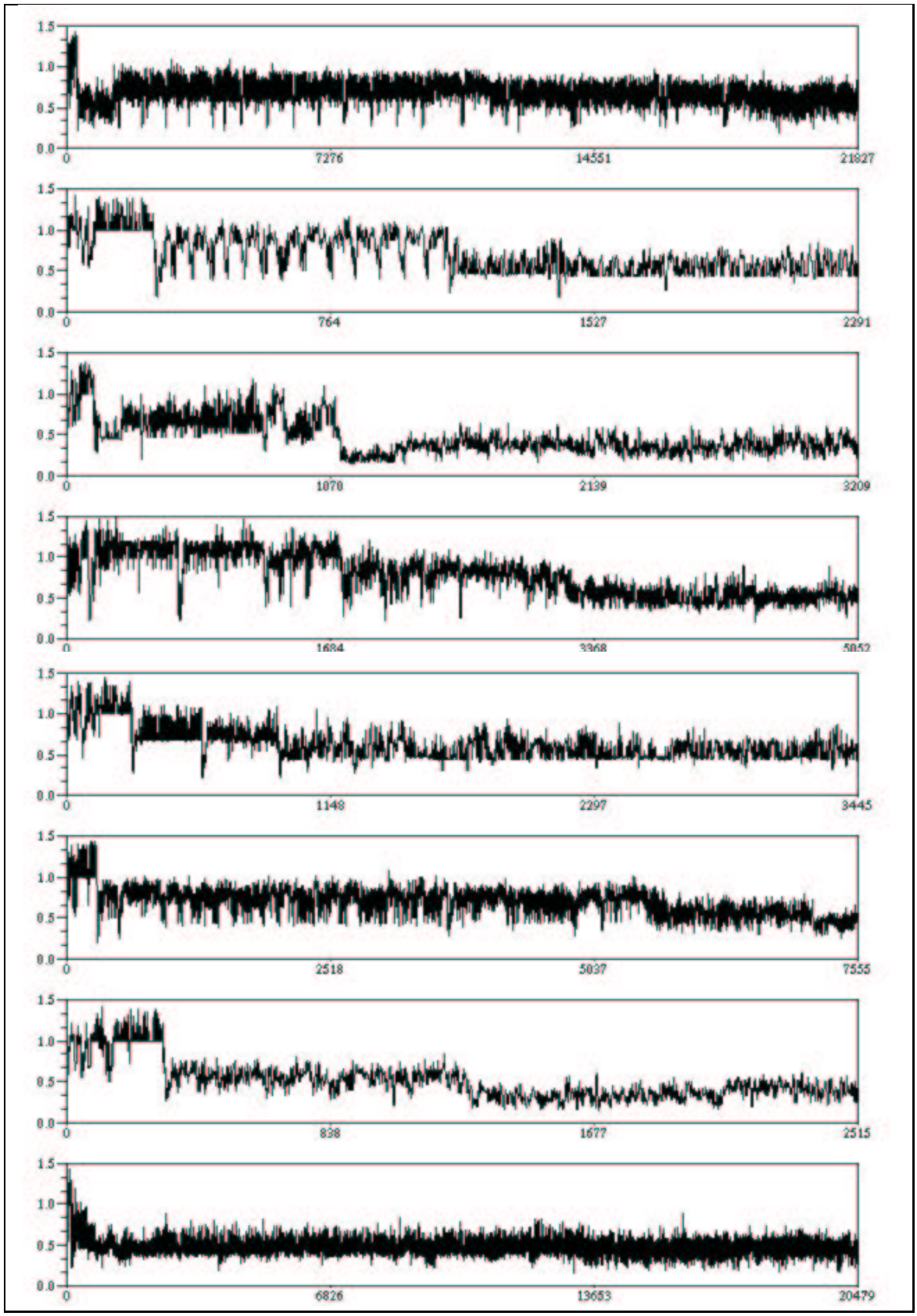

Figure 5: Multidimensional Hölder exponent for eight different sets of crash data.

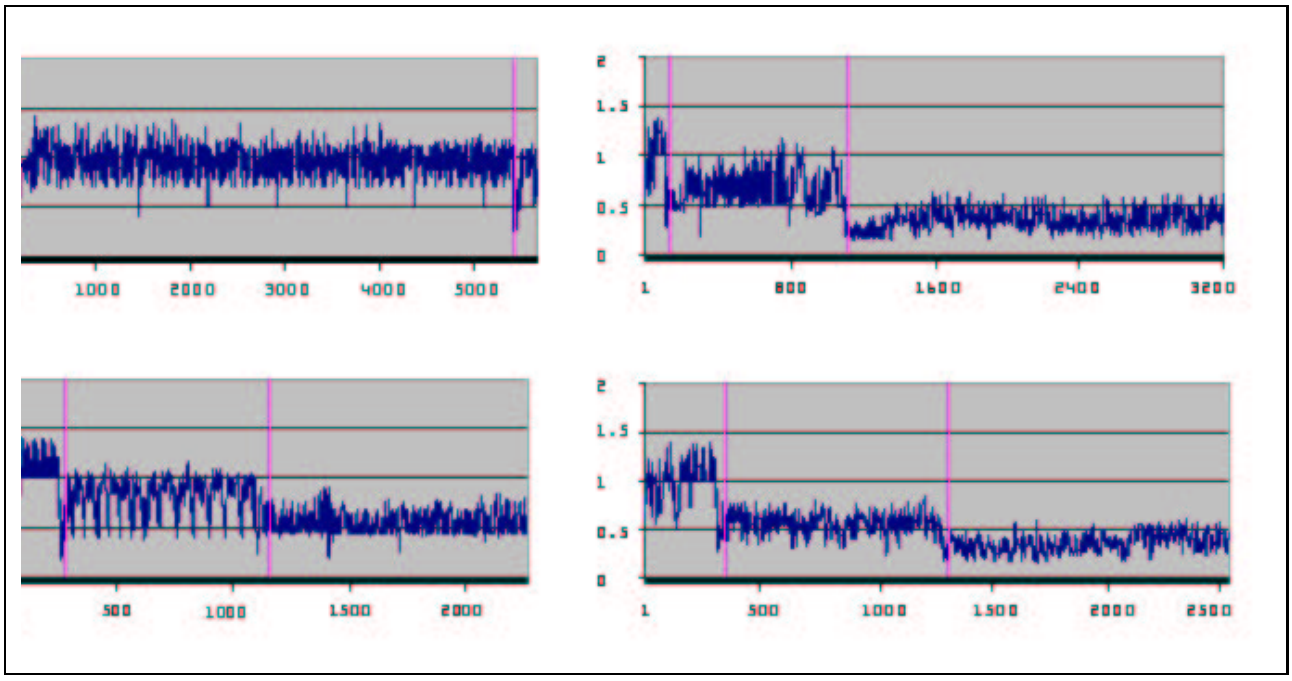

Figure 6: Four Hölder exponent time series ending in crash with pink dashes showing fractal breakdowns detected using modified Shewhart algorithm.

References

[1] Michael Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[2] Khalid Daoudi and Jacques Lévy Véhel. Speech signal modeling based on local regularity analysis. *IASTED IEEE International Conference on Signal and Image Processing*, 1995.

[3] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications.* John Wiley and Sons, 1990.

[4] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[5] E. Tosan J. Thollot, C.E. Zair and D. Vandorpe. Modeling fractal shapes using generalizations of IFS techniques. In *Fractals in Engineering*, pages 65–80. Springer, 1997.

[6] K. Trivedi K. Vaidyanathan. A measurement-based model for estimation of resource exhaustion in operational software systems. *Proceedings of the 19th International Symposium on Software Reliabiltiy Engineering*, 1998.

[7] Khalid Daoudi, Jacques Lévy Véhel and Yves Meyer. Construction of continuous functions with prescribed local regularity. *Journal of Constructive Approximation*, 14(3):349–385, 1998.

[8] Benoit Mandelbrot Laurent Calvert, Adlai Fisher. Large deviations and the distribution of price changes. Technical Report 1165, Cowles Foundation, Sep 1997.

[9] Benoit Mandelbrot. *Fractals: Form, Chance, and Dimension.* W.H. Freemen and Company, 1977.

[10] Petros Maragos. Modulation and fractal models for speech analysis and recognition. *Proceedings of COST-249 Meeting*, Feb 1998.

[11] Igor V. Nikiforov Michele Basseville. *Detection of Abrupt Changes : Theory and Application.* Prentice Hall, April 1993.

[12] R. Morin A.L. Goldberger L.A. Lipsit N. Iyengar, C.K. Peng. Age-related alterations in the fractal scaling of cardiac interbeat interval dynamics. *American Journal of Physiology*, 40(4):1078–1084, 1996.

[13] Z. R. Struzik. Revealing Local Variablity Properties of Human Heartbeat Intervals with the Local Effective H?lder Exponent. *Technical Report INS-R0015, CWI, Amsterdam, The Netherlands*, July 2000.

[14] David Parnas. Software aging. *Proceedings of the 16th Intl. Conference on Software Engineering*, pages 279–287, 1994.

[15] Edgar E. Peters. *Fractal Market Analysis.* John Wiley and Sons, 1994.

[16] A. Van Moorsel S. Garg. Towards performability modeling of software rejuvenation.

[17] K. Vaidyanathan K. Trivedi S. Garg, A. Van Moorsel. A methodology for detection and estimation of software aging. *Proceedings of the 9th International Symposium on Software Reliability Engineering*, pages 282–292, 1998. Paderborn, Germany.

[18] K. Trivedi T. Dohi, K. Goševa-Popstojanova. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. *Proceedings of the 2000 Pacific Rim International Symposium on Dependable Computing*, pages 77–84, 2000.

[19] P. Heidelberger S. W. Hunter K. S. Trivedi K. Vaidyanathan W.P. Zeggert V. Castelli, R.E. Harper. Proactive management of software aging. *IBM J. Res. and Dev.*, 45(2):311–332, Mar 2001.

[20] Jacques Lévy Véhel and Khalid Daoudi. Generalized IFS for signal processing. *IEEE DSP Workshop*, Sep 1996.

[21] Walter Willinger Daniel Wilson Will Leland, Murad Taqqu. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

[22] N. Kolettis N. D. Fulton Y. Huang, C. Kintala. Software rejuvenation: Analysis, module and applications. *Proceedings of the 25th Intl. Symposium on Fault-Tolerant Computing*, 1995.

[23] S. Jaffard, Multifractal formalism for functions, *Comptes Rendus de l'Acadmie des Sciences,*, 317: 745-750, 1993.

Appendix 1: Algorithm for Numerical Evaluation of Hölder Exponent of a Discrete Signal

Suppose $\{y_0, y_1, y_2, ..., y_n\}$ is a time series of length n representing some quantity measured at uniform intervals. The idea is that the degree to which the function is smooth or chaotic (and therefore the strength of its Hölder exponent) at a certain data point y_i is a function of a number of data points preceding and following it. This number, s, is called the window width and is not fixed but is rather a parameter set by the user when running the algorithm. The weight that each of the $2s$ values (s values on each side of the data point) has on the ultimate calculation of the Hölder exponent is controlled by another parameter, λ. λ is called the weighted regression coefficient and must satisfy $0 < \lambda < 1$. Adjusting the strength of λ will allow us to adjust the degree to which the data points further away from the point we are interested in influence the calculation of the Hölder exponent at that point. Obviously nearby points should have a greater weight than distant points.

To calculate the Hölder exponent at the ith point in a data set, we begin by choosing a window width, s, and a value for λ. We have used $\lambda = 0.5$ and $s = 10$ for all the Hölder exponent calculations appearing in this study.

Next, we calculate a value, $R_{i,k}$, for each integer $k \neq 0$ from $-s$ to s in the following manner:

$$R_{i,k} = \frac{\log |y_{i+k} - y_i|}{\log(\frac{|k|}{1000})}$$

In the above case, k must satisfy $0 \leq i + k \leq n$. For data points at the very beginning or end of the time series for which s values to the left or right are not available, s is shrunk to the appropriate size.

Then, in the second step, for each integer j, $1 \leq j \leq k$, calculate

$$h_{i,j} = \min\{R_{i,k} : |k| \leq j\}$$

(This corresponds to taking the lim inf in the equation for the Hölder exponent).

In the third step, calculate $\mathcal{H}_i$, the estimate of the Hölder exponent at the ith data point by computing the weighted average of the approximations $h_{i,j}$:

$$\mathcal{H}_i = \frac{1-\lambda}{1-\lambda^k}(h_{i,1} + \lambda h_{i,2} + \lambda^2 h_{i,3} + ... + \lambda^{k-1} h_{i,k})$$

where more weight is given to terms with a smaller k (i.e., h closer to 0 in equation 1).

Appendix 2: Shewhart Algorithm Adopted for Online Detectionion of a Fractal Breakdown

Though numerous good change detection algorithms exist, most of them presuppose the a priori knowledge of the main parameters of the post-change signal. This is not the case in our situation where the change detection needs to be performed in real time. We used the classical *Shewhart control charts* algorithm (see e.g. [11]), however, for the reasons mentioned above, we needed to modify it so that, unlike the classical, situation, the mean and the variance of the signal are estimated on-line instead of being known *a priori*. The Shewhart control charts is based on the maximum likelihood principle where the change hypothesis of is tested against the no change hypothesis. Between the changes the signal is assumed to be stationary independent Gaussian process with constant mean and variance, where we are trying to detect the change in the mean value considered as a parameter.

Let y_i, $i = 1, 2, , T$ be a noisy time series which is observed in real time and in which we want to detect a moment of sharp change. Here we are interested only in downward changes (drops), since this is the type of change we want to be able to detect in the Hölder exponent signal (we hypothesize that the 2nd such change indicates a dangerous level of resource exhaustion in the system).

Fix the starting point by setting $T_0 = 0$. Skip first N data points; then for each data point with $n > T_0 + N$ compute the statistical estimate of the mean and the standard deviation of the current fragment

of our time series:

$$\mu_n = \frac{1}{n - T_0} \sum_{j=T_0+1}^{n} y_j;$$

$$\sigma_n = \sqrt{\frac{1}{n - T_0 - 1}} \sum_{j=T_0+1}^{n} (y_j - \mu)^2.$$

Compute the local (last N points) average of the signal:

$$a_n = \frac{1}{N} \sum_{j=n-N+1}^{n} y_j.$$

Then compute the normalized discrepancy term between the local (moving) average and the global mean:

$$d_n = \frac{\sqrt{N}}{\sigma_n}(\mu_n - a_n).$$

Check if the discrepancy d_n exceeds threshold ξ, i.e. if the inequality

$$d_n > \xi$$

holds. Before declaring that a change has occurred we are looking to confirm that the change is sustained and not merely an isolated local anomaly. If the inequality $d_n > \xi$ holds for p consecutive points (n, n+1,, n+p-1) we decide that a (sustained) downward change and restate the starting point for our parameter estimation by setting

$$T_0 = n.$$

The parameters: N (local averaging time); p (change confirmation time); ξ (deviation threshold) are controllable and can be tuned to make the algorithm sufficiently sensitive but not too prone to false alarms.